High-Dimensional Metamodeling for Prediction of Clock Tree Synthesis Outcomes

Andrew B. Kahng[+†], Bill Lin[†] and Siddhartha Nath[+]
[+]CSE and [†]ECE Departments, University of California at San Diego
{abk, billlin, sinath}@ucsd.edu

***Abstract*—Clock tree synthesis (CTS) is a key aspect of on-chip interconnect, and major consumer of IC power and physical design resources. In modern sub-28nm tools and flows, it has become exceptionally difficult to satisfy skew, insertion delay and transition time constraints within power and area budgets, in part because commercial tools (with their many knobs) have become highly complex. This complexity, along with the complicated structure of real-world CTS instances (hierarchy, dividers, etc.) and floorplan contexts (aspect ratios, obstacles, etc.) make it very difficult to predict skew, power and other important metrics of CTS outcomes. In this work, we study CTS estimation in the high-dimensional parameter space of instance constraints and floorplan contexts. Using two leading commercial CTS tools as our testbed, we develop predictors, classifiers and "field of use" characterizations that can enable IC design teams to achieve required CTS solution quality through understanding of appropriate parameter subspaces. Our hierarchical hybrid surrogate modeling approach mitigates challenges of parameter multicollinearity in high dimensions. It achieves, e.g., worst-case estimation errors of 13% in contrast to 30% errors in [17]. We demonstrate use of a 94%-accurate "oracle" classifier and estimation models to predictably achieve CTS outcomes that meet specified constraints and target metrics.**

I. Introduction

On-chip clock distribution solutions are increasingly critical to achieving IC power, area and design quality goals. For high-end SOC designs in advanced nodes, physical implementation teams must explore a wide range of different clock tree synthesis (CTS) styles and methodologies in order to deliver required performance while minimizing power and resource costs. Because SOCs contain multiple heterogeneous blocks, different flows or tool settings may be preferable for different blocks.[1] However, selection of tools and tool options remains fairly arbitrary, due to limited understanding of the *field of use* ("sweet spot") of individual tools, as well as the high complexity and runtimes of the tools themselves. Thus, users face a growing inability to predict CTS outcomes, and a higher incidence of failed runs. This has unfortunate implications with respect to design turnaround time and tool license cost.

In an ideal world, EDA tool users would be empowered by "oracle" estimators and classifiers that could answer such fundamental questions as: "What will be the outcome (power, area, skew, etc.) of this run?"; "Which tool is best suited for this design instance?"; or "How should I set tool options and parameters to obtain best possible results on this design instance?" Such capabilities would enable more comprehensive design space exploration, faster design closure, and better silicon QOR. Our present work seeks to develop such estimators and classifiers for the CTS context. In doing so, we come to grips with two fundamental challenges that previous works leave unaddressed: (1) building predictive models in very high-dimensional parameter spaces; and (2) identifying the parameters of layout context and clock structure that make CTS outcomes so difficult to predict.

The difficulty of CTS prediction begins with the wide range of metrics according to which clock trees must be evaluated: wirelength, buffer area, insertion delay, skew, power, etc. Then, each of these metrics is the end result of the user's setting of many constraints and options in a highly-complex, black-box tool: maximum buffer and sink transition times, non-default routing rules, maximum insertion delay, clock buffer and inverter library cell sizes, maximum number of levels in the clock tree, maximum skew, etc. This results in a very high-dimensional estimation and modeling task.

Estimation of CTS metrics has been previously addressed in the METRICS research of Kahng and Mantik [14]; more recently, Kahng et al. [17] apply metamodeling techniques as the basis of accurate CTS estimates. Both of these previous works make simplifications that diverge from the reality of clock tree designs. For example, both assume that CTS instances arise in rectangular blocks; [17] further assumes that clock sinks are uniformly placed when fitting predictive models. On the other hand, our present studies indicate that floorplan context is essential to include in any practical CTS model. Figures 1(a) and (b) illustrate that fall delay may vary up to 43% for different locations of clock entry points with a given fixed block aspect ratio, as highlighted by the red ovals. The red dotted lines show that power as well as fall delay vary by up to 45% when the block aspect ratio is varied. When block aspect ratio, location of clock entry point, or other necessary parameters of the CTS instance (e.g., nonuniformity of sink placement) are identified and added into the model development, the resulting high parameter dimensionality and parameter *multicollinearity* (discussed below) prove challenging for previous methods.

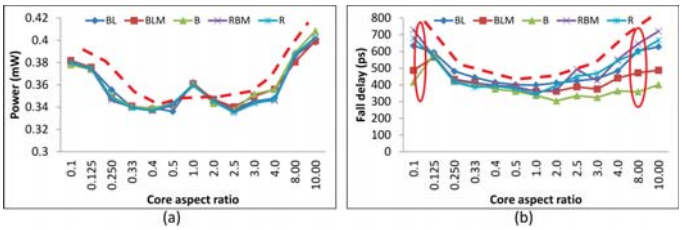

Fig. 1. CTS metrics for 3K sinks, with different clock entry points (cf. Figure 2) and core aspect ratios. (a) Power. (b) Fall delay.

CTS research also faces the challenge of using real-world testcases. Instances used in recent academic research, from the "r1" $\sim$ "r5" testcases introduced by Tsay [22], to the testcases at the 2010 *International Symposium on Physical Design* [27] CTS contest, omit clock dividers, clock gating cells, clock MUXes, glitch-free clock switchers and other structural elements that are expected in a complex, low-power SOC design. The ability to comprehend such testcase characteristics is essential to accurate estimation with realistic problem instances. However, this again expands the parameter space and makes the modeling problem much more difficult.

In this work, we make progress from several directions toward the goal of realistic, practical CTS estimation. (1) We identify parameters that succinctly capture complex characteristics of realistic CTS testcases, such as sink placement nonuniformity or irregularity of hierarchical insertion delays. (2) We also propose a divide-and-conquer approach that achieves decomposition of high-dimensional

[1]Examples of commercial tools that perform CTS include Cadence *SOC Encounter vEDI10.1* [31], Synopsys *IC Compiler vG-2012.06-SP3* [33], Mentor Olympus-SoC [28] and ATopTech Aprisa [25].

modeling problems into lower-dimensional models. Specifically, our new *hierarchical hybrid surrogate modeling* (HHSM) approach addresses the long-standing issue of large estimation errors in higher dimensions due to multicollinearity. (3) We show the practicality of developing an "oracle" classifier that predicts which commercial CTS tool will perform better, according to specified metrics of interest, on a given problem instance. (4) We also identify the proper parameter settings for commercial tools, by estimating the subspace of the tool parameter space in which the tool will successfully optimize a specified metric while satisfying all given solution constraints. (5) Of independent interest is our means of generating CTS testcases with realistic structure, going beyond the 2010 ISPD contest testcases. Having realistic CTS instances not only addresses a long-standing gap in the CTS research literature, but provides a basis for model development that allows us to identify more relevant (for prediction of CTS outcomes) instance parameters.

We summarize our key contributions as follows.

- We are the first to identify and study *difficult*, layout context-derived dimensions of CTS prediction: block aspect ratio, nonuniform sink placement, and non-rectangular block shape. We formulate an appropriate set of modeling parameters to capture these complex characteristics, and use high-dimensional metamodeling to solve practically useful CTS prediction problems.
- To conquer the problem of multicollinearity, we propose *hierarchical hybrid surrogate modeling* (HHSM) for decomposition of high-dimensional problems into lower-dimensional models. We demonstrate with complex and realistic testcases that HHSM can reduce the worst-case estimation error to 13%, which is a 57% reduction of error in comparison with previous metamodeling techniques [17].
- Starting from an understanding of practical use models for CTS prediction and estimation, we show the feasibility of answering with high accuracy three questions that design teams are always asking:
  - Which tool should be used?
  - How should the tool be driven?
  - How wrong can the model guidance be?
- We develop a means of generating realistic CTS testcases that improve upon 2010 ISPD contest testcases by adding clock structures present in real-world CTS instances.
- We empirically demonstrate that our prediction methodologies can be applied to new CTS problem instances to obtain a target value of a given metric (skew $\leq$ 30ps).

The remainder of this paper is organized as follows. Section II summarizes prior related work. Section III describes our problem formulation, testcase generation and metamodeling flows. Section IV describes our HHSM methodology to break the curse of multicollinearity, and metrics estimation results with HHSM. Section V provides our answers to questions about practical use models for prediction/estimation, and Section V-B demonstrates the application of our methodologies to a design flow using a new testcase and commercial tools to achieve a specified metric. We outline future work and conclude in Section VI.

II. Related Works

Prediction of clock network metrics is a sparsely explored area in CAD. Kahng et al. [14] use their METRICS infrastructure to collect design flow information during clock network synthesis, and perform data mining using *CUBIST* to estimate clock skew and insertion delays. They use *CTGen*, a commercial CTS tool in 2001, to conduct their studies on industrial testcases and report correlation coefficients of around 0.82 in predicting maximum and minimum insertion delay, maximum skew, and routing violations. However, they do not report errors in estimating individual metrics. Samanta et al. [21] use Support Vector Machine (SVM) [23] regression to estimate clock skew with an accurate delay model to size buffers and wires in a non-tree clock network. They do not show parameter subspaces within which maximum skew constraints are satisfied, nor the efficiency of tools. Kahng et al. [17] demonstrate several metamodeling techniques for estimation of wirelength and buffer area in a high-dimensional parameter space. However, all of their results are obtained with 10 parameters and uniformly placed sinks in a rectangular block with aspect ratio set to one. As we show below, realistic CTS instances have attributes that present much more difficult modeling challenges.

Various arenas of IC design use *surrogate models* for early design space exploration and characterization of the parameter space. These models are used for either (1) classification or (2) estimation. Callegari et al. [3] use classification to develop a feature-based rule learning algorithm and apply it to analyze silicon test measurement data. They uncover mismatches between design and silicon even in the presence of random noise. Chakrabartty et al. [6] implement an analog system-on-chip 24-class SVM classifier [34] for biometric signature verification by analysis of voice samples. Estimation models are either Gaussian process-based (e.g., Radial Basis Functions (RBF) [36] and Kriging (KG) [19]) or tree-based (e.g., Multivariate Adaptive Regression Splines (MARS) [35]). RBF uses kernel functions that are symmetric and centered at each parameter, whereas KG is an interpolation technique that models random noise by using a weighted correlation function. MARS is an additive tree-based regression model that uses piecewise splines to fit training data. A more detailed review of these methods is given in [10]. Ilumoka [12] estimates crosstalk in interconnects by training a RBF model with SPICE simulations. Liu [18] models on-chip temperature and IR drop by using KG. Dubois et al. [8] use KG to estimate area of network-on-chip (NoC) routers. Kahng et al. [16] use MARS to estimate NoC router area and power. Goel et al. [9] demonstrate that *weighted surrogate modeling* is significantly more accurate than individual surrogate models. A very recent work of Kahng et al. [17] devises *Hybrid Surrogate Modeling* (HSM), an extension to the model proposed by [9]. The authors demonstrate the advantages of using Adaptive Sampling (AS) [7] over Latin Hypercube Sampling [13] and report reduction of worst-case errors by up to 3$\times$ with HSM and AS.

III. Design of Experiments

We now describe our problem formulation, our testcase generation methodology, and our metamodeling flows.

A. Problem Formulation

We formulate CTS metrics prediction as a high-dimensional metamodeling problem. To this end, we choose appropriate modeling parameters that capture the complex characteristics of CTS problem instances and tool knobs. Table I defines these parameters and the ranges of values they typically take. M_{sink} is an architectural parameter; M_{core}, M_{AR}, M_{CEP}, and M_{block} are floorplanning parameters. Figure 2 shows five possible clock entry points, with M_{CEP} calculated as the Manhattan distance to a given clock entry point (CEP) from $BL = (0,0)$. M_{skew}, M_{delay}, $M_{buftran}$, $M_{sinktran}$, M_{FO}, $M_{bufsize}$, and M_{wire} are physical design constraints used as tool knobs. M_{DCT} expresses the nonuniformity of sink placement and enables us to distinguish between different nonuniform placements; details of the M_{DCT} calculation are given in Section III-B. We denote the number of modeling parameters as D. Overall, we have 13 modeling parameters.

B. Testcase Generation

We generate four templates of artificial testcases (A1, A2, A3, A4) and two templates of realistic testcases (R1, R2). Artificial testcases

TABLE I
MODELING PARAMETERS AND THEIR RANGES

Parameter	Description	Range
M_{sink}	# sinks	{1, 3, 10, 30}K
M_{skew}	max skew	[1, 500]ps
M_{delay}	max insertion delay	[0.1, 5.0]ns
M_{core}	block area	[2, 40]mm^2
M_{AR}	core aspect ratio (AR)	[0.125, 8.0]
M_{CEP}	clock entry point	{BL, BLM, B, RBM, R}
M_{block}	blockage as % of block area	[0, 60]%
M_{DCT}	nonuniformity measure	-
$M_{buftran}$	max buffer transition	[200, 1000]ps
$M_{sinktran}$	max sink transition	[200, 600]ps
M_{FO}	max fanout	[8, 128]
$M_{bufsize}$	max buffer size	x[8, 24]
M_{wire}	max wire width (in NDR)	x[1, 3]

contain only clock sinks (i.e., no combinational logic), whereas realistic testcases contain combinational logic and common clock structures such as dividers, MUXes, clock-gating cells (CGCs), delay shifters, and glitch-free clock switchers, to create hierarchical structure in the clock tree. Each of these templates can be placed in rectangular (core aspect ratio $= 1$) or non-rectangular (core aspect ratio $\neq 1$) blocks, and CEP can be placed at different locations of the block as shown in Figure 2. Our testcases and contexts improve upon the well-known 2010 ISPD CTS benchmarks [27] that use only buffers and inverters to specify a clock tree hierarchy, specify cores only with aspect ratio $= 1$, and use only placement blockages.

Our four artificial testcase templates are

A1 – sinks placed uniformly in a rectangular block;
A2 – sinks placed uniformly in a non-rectangular block;
A3 – sinks placed nonuniformly in a rectangular block; and
A4 – sinks placed nonuniformly in a non-rectangular block.

Template A1 is generated with custom *Tcl* scripts. Sinks are placed in rows with equal whitespace between each pair of adjacent sinks. **Template A2** is generated by creating placement and routing blockages with configurable dimensions and locations within the layout region. After creating the blockages, sinks are placed uniformly in the non-rectangular block. **Template A3** is generated by initially placing $1.6\times$ the required number of sinks uniformly within a rectangular block. The excess sinks are removed by repeating the following four steps.

1) Apply Box-Muller transformation [2] to obtain a bi-variate Gaussian distribution, with zero mean and unit variance.
2) Choose a sink s_c at random and designate it as the *center*, and choose M closest sinks to s_c.[2]
3) Obtain probabilities of these M sinks using Box-Muller transformation. Values of x_1 and x_2 in the transform are respectively computed as horizontal and vertical distances of a sink from s_c.
4) Remove a sink if its probability is $\geq 34\%$.

To quantify nonuniformity, we (1) divide the block area into $m \times n$ 2-D grids, where m and n are integers,[3] (2) compute the χ^2 value of each grid as $\chi^2_j = \sum_i \frac{(n_i - \hat{n})^2}{\hat{n}}$, where $j = 1, ..., \#\text{grids}$, n_i is the Manhattan distance between the i^{th} pair of sinks in a grid, and $\hat{n}$ is the average distance between sinks in a uniform placement, and (3) apply discrete cosine transform (DCT) on the χ^2 values across all the grids. Having obtained the DCT coefficients corresponding to each grid, we use the sum of the magnitudes of these coefficients as our measure of nonuniformity (when the placement is uniform, the sum of the DCT coefficients is zero). **Template A4** is generated by creating a non-rectangular block using the methodology to generate template A2 and then performing nonuniform sink placement using the methodology to generate template A3.

To generate realistic testcases, we construct two templates, R1 and R2, in *Verilog* RTL that represent different clock hierarchies using common clock tree structures [26], [30]. At each level of the clock tree, the number of sinks is an integer divisor of M_{sink}, such that the sum of sinks across all levels in the tree is equal to M_{sink}. Figures 3(a) and (b) respectively show templates R1 and R2. **Template R1** is generated by cascading three dividers and a glitch-free MUX, and inserting CGCs before each sink group, K_1–K_6. One CGC is inserted after the clock root pin to facilitate clock-gating of the entire tree. Each CGC has its own enable signal. **Template R2** is generated by inserting CGCs before each clock divider and using a combination of MUXes and glitch-free clock switchers to deliver clock to sink groups K_1, K_2 and K_3. These templates demonstrate that sink groups can be at different hierarchies in a clock tree, as well as the presence of reconvergent paths from the clock source to the sinks.

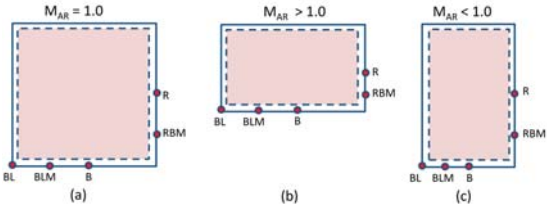

Fig. 2. Five clock entry points for M_{AR}: (a) $= 1.0$, (b) > 1.0 and (c) < 1.0.

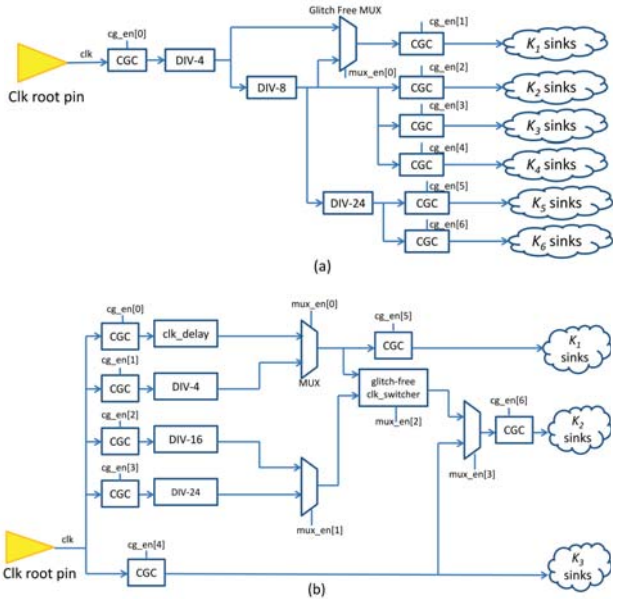

Fig. 3. Realistic templates: (a) R1; (b) R2.

C. Modeling Flow

We generate a total of 4500 golden data points by running several instances of CTS on the templates described in Section III-B and by choosing values of the parameters shown in Table I. Each of these 4500 data points is the average of five CTS runs per released version of each commercial tool in light of tool noise as studied in [15]. With larger numbers of sinks (e.g., 10K, 30K) each CTS run may take up to 80 minutes for each commercial CTS tool. Because it is expensive to generate data points, we use adaptive sampling (AS) to optimally select the minimum number of samples based on exploration and exploitation strategies [17]. We use AS to generate training set sizes with {36, 48, 56, 84, 116, 148, 196, 224} data points for $D = \{8, 9, 10, 11, 12, 13\}$. The remaining data points are used to test the

[2] $M = 500$ in our experiments.

[3] We use grid sizes of eight rows and one-eighth block width in our experiments. It is possible to use any other reasonable values, e.g., square grids of dimensions $20\mu m \times 20\mu m$.

accuracy of our models. We use academic *MATLAB* [34] toolboxes for MARS, RBF, and KG, and the built-in *MATLAB* toolbox for SVM; we develop our own code for HSM [17]. Table II shows the configuration options for the *MATLAB* toolboxes. The runtime for sampling and model derivation with 224 data points and 13 model parameters is at most 28 minutes on a 2.5GHz Intel Xeon system, and at most 16 minutes with eight parameters.

Figure 4 shows our tool flow in detail. We implement the templates in *Verilog* RTL, and then perform synthesis using Synopsys *Design Compiler vG-2012.06* [32] with TSMC45GS and TSMC65GPLUS technology libraries to generate a gate-level netlist. We use the modeling parameters to generate a placed Design Exchange Format (DEF) file. Then, we use the placed DEF along with the remaining parameters as CTS constraints to generate a CTS problem instance. This instance is given as input to two market-leading commercial CTS tools, *ToolA* (June 2012 release) and *ToolB* (December 2010 release), to perform synthesis and routing of the clock tree. Lastly, all the metrics of interest are extracted using custom *Tcl* and shell scripts to generate the golden data points. We develop separate models for artificial and realistic testcases at 45nm and 65nm respectively, and report the average of prediction errors unless otherwise specified.

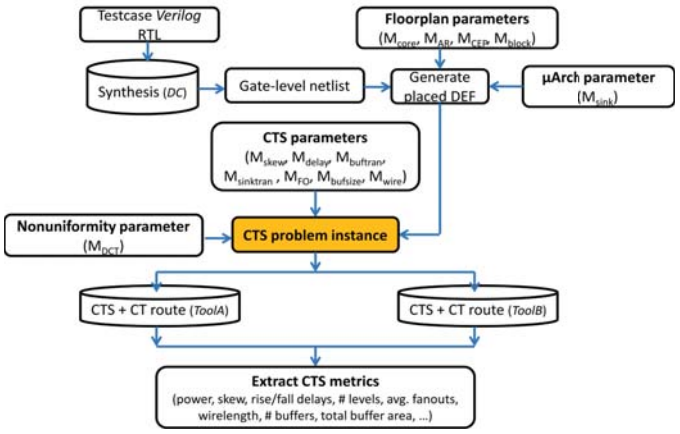

Fig. 4. Our CTS tool flow.

TABLE II
METAMODELING TOOL CONFIGURABLE PARAMETERS

Metamodel	Tool	Value
MARS	*ARESLAB*	max basis functions = {100} max interactions = {12} spline type = {cubic}
RBF	*RBF2*	kernel type ={multiquadrics} $4.5 \leq radius \leq 0.5$ method = {ridge regression}
KG	*DACE*	reg. model = {order 1 and 2 poly} corr. model = {EXP} θ = {20, 40}
SVM	*MATLAB v2012b*	kernel = {RBF} method = {QP}

IV. OVERCOMING THE CURSE OF HIGH-DIMENSIONALITY IN METRICS ESTIMATION

We reproduce the approaches of [17] and [14] as baselines for our studies, and accordingly use *CUBIST*, MARS, RBF, KG, and HSM to derive surrogate models by varying D. We use AS for MARS, RBF, KG, and HSM techniques.

Large estimation errors in previous techniques as D increases

Figures 5(a) and (b) show the maximum absolute percentage errors (MAPE) in skew and delay estimation respectively for MARS, RBF, KG, and HSM with different values of D. We observe that errors increase across all surrogate models; errors are >100% for MARS, RBF, and KG as D increases [17]. HSM performs the best among these with MAPE around 36% when $D = 13$. We observe similar values of MAPE in power and wirelength estimation of the clock tree as shown in Figures 6(a) and (b).

We also compare accuracy of MARS in skew and delay estimation because both are piecewise additive models. We observe that when $D = \{8,9\}$, the estimation errors of *CUBIST* and MARS are similar; however, when $10 \leq D \leq 13$, MARS is 32% more accurate than *CUBIST*. MARS is more accurate because it uses cubic splines, whereas *CUBIST* uses a linear additive approach that does not scale at high dimensions. In general, previous techniques have large estimation errors and cannot reduce MAPE below 36%.

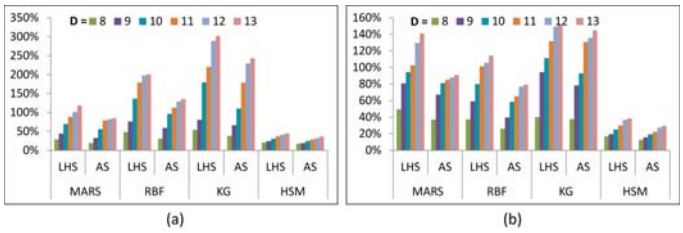

Fig. 5. Maximum absolute % errors in (a) skew and (b) delay estimation.

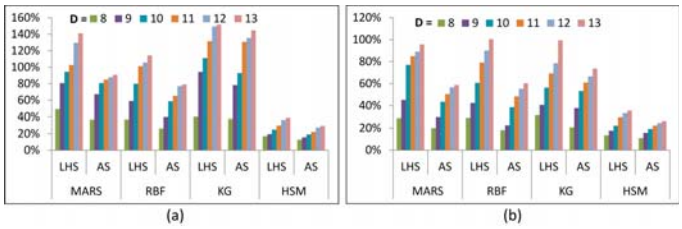

Fig. 6. Maximum absolute % errors in (a) power and (b) wirelength estimation.

Reasons for large estimation errors at high dimensions

Multicollinearity [20] is an important reason why previous techniques incur large estimation errors at high dimensions. Multicollinearity arises when parameters are linearly dependent. It causes the matrix of parameters to be ill-conditioned and have less than full column rank. The regression coefficients become highly sensitive to small changes in parameter values and make the surrogate models inaccurate. In IC design problems we cannot drop parameters to avoid multicollinearity because we need to understand interactions between them. For example, M_{AR} and $M_{buftran}$ may be linearly related, but each of these parameters affects skew differently. Furthermore, deriving accurate models becomes difficult as D increases because the parameter space grows exponentially with respect to D [24]. For these reasons, we observe large estimation errors when $D > 10$ in Figures 5 and 6.

Our approach: Hierarchical hybrid surrogate modeling

To break the "curse of high dimensionality" we propose a new method, *hierarchical hybrid surrogate modeling* (HHSM). HHSM uses a divide-and-conquer approach, dividing the modeling parameters into two groups: one group uses k parameters that exhibit low multicollinearity, and the other group uses the remaining $D-k$ parameters that may exhibit large linear dependence with the other parameters. In the conquer step, models from these two groups are combined using weights determined by least-squares regression (LSQR).

Due to practical limitations of generating training data points (i.e., each data point is very expensive to generate), the training and test sets are typically smaller than the required exponential number of data points in a high-dimensional parameter space with dimension D.

Therefore, a model with $k \leq D$ independent and identically distributed (i.i.d) parameters is a good approximation of the limited number of data points. Formally, HHSM is given by

$$\hat{y}(\vec{x}) = m_0 + m_1 \cdot HSM_{1..k} + m_2 \cdot HSM_{k+1..D} \quad (1)$$

where m_1 and m_2 are weights of estimated responses of HSM of parameters 1 through k and $k+1$ through D, respectively, and m_0 is the bias. To determine these weights, we use LSQR to fit the model to a randomly selected 60% of (training) data points. We compute the *generalized cross-validation* (GCV) error [10] on the remaining 40% of the data points.

Accuracy of the HHSM model is sensitive to the choice of parameters used to derive each HSM model. We use *variance inflation factor* (VIF) [1] as a measure of parameter compatibility. Our objective is to minimize the error of the HSM model with the first k parameters. We observe that HSM is highly accurate when $k = 6$, and when these parameters are more or less i.i.d. We determine these six out of $D > 6$ parameters using the following steps.

1) Choose six parameters with minimum sum of VIF values.[4]
2) Derive one HSM model with these six parameters.
3) Derive another HSM model with the remaining parameters. This model may have larger estimation errors than the model in Step 2 because the VIF values of these parameters may be large [17], thereby demonstrating effects of multicollinearity.
4) Determine values of m_0, m_1, and m_2 using LSQR when combining the two HSM models.

HHSM is able to "cure" multicollinearity effects when $10 \leq D \leq 13$. Figures 7(a) and (b) respectively compare the maximum and average percentage estimation errors between HSM and HHSM models. We observe that across all CTS metrics, MAPE for HHSM is around 13%, which is a 57% reduction of error in comparison to the previous metamodeling techniques [17]. The errors are almost flat when $8 \leq D \leq 13$. When parameters exhibit low multicollinearity, then a small value of D may have larger estimation errors with HHSM as compared to a large value of D, due to the presence of the bias term, m_0, in Equation (1).

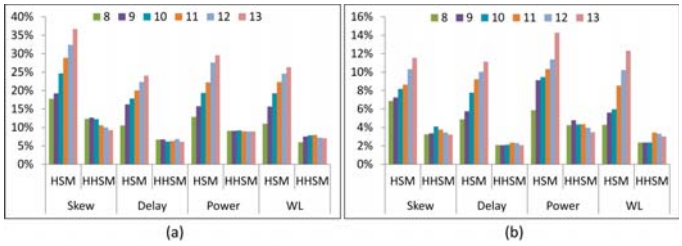

Fig. 7. Comparison of (a) maximum and (b) average percentage errors in skew, delay, power, and wirelength estimations between HSM and HHSM.

V. A Practical Application of Prediction for Clock Trees

IC designers [26], [29], [30] often want to know answers to questions on practical use models of commercial tools. We now describe use of an "oracle" classifier and estimators from Section IV to answer three relevant questions. In Section V-B, we empirically demonstrate the practical application of our prediction methodologies to a new CTS problem instance, implemented with commercial tools, to obtain a specified skew target.

[4]When the VIF values are more than 0.5 for more than two parameters, we observe that the maximum error of the HHSM model can be up to 45%.

A. Addressing CTS Questions for IC Designers

We have explored new methodologies using classifiers and estimators to answer three questions often posed by IC designers.

Question 1: Which Tool Should Be Used? To answer the question of which tool will perform better for a given CTS problem instance and a specified metric, we conduct two experiments: (1) develop a two-class "oracle" classifier with the SVM toolbox in *MATLAB*, and (2) use estimators to determine the better tool.

In the first experiment, we construct classifiers for four metrics, namely, skew, clock power, delay and clock tree wirelength. If a tool satisfies the maximum skew and insertion delay constraints, it is assigned a score of "+1", else its score is "−1". For power and wirelength, a tool's score is "+1" when both maximum skew and delay constraints are satisfied and it produces the minimum value of power or wirelength for that problem instance. Figure 8 shows the maximum percentage error in classifying a tool that will perform better. The error is calculated as a percentage of mispredictions, that is, the ratio of the number of incorrect results to the total number of problem instances in the test set multiplied by 100. We observe that when $D < 10$, our classifier predicts correctly 95% of the time. At higher D, the accuracy degrades to around 94% because of multicollinearity.

D	Skew	Power	Delay	Wirelength
8	5.26	4.55	4.87	4.92
9	5.26	4.6	4.93	5.01
10	5.82	4.62	4.94	5.03
11	5.88	4.9	4.94	5.11
12	6.12	5.23	4.95	5.25
13	6.13	5.23	4.98	5.27

Fig. 8. Classification error % for the question of which tool will perform better.

In the second experiment, we derive HHSM models for *ToolA* and *ToolB* for skew, delay, power and wirelength. Then, for each CTS instance in the test set, we use the HHSM model of each tool to estimate the metric, compare the values, and choose a winner. We compare our prediction against post-CTS data-based selection of the winner for skew, delay, power and wirelength. Prediction error occurs if post-CTS data-based prediction does not match the HHSM-based prediction. Figure 9 shows that errors are slightly higher than in Figure 8: for some observations, *ToolA* is better than *ToolB* by ∼8%, but HHSM incurs a prediction error of ∼10% and predicts *ToolB* to be better than *ToolA*. However, we observe the results are comparable in general.

D	Skew	Power	Delay	Wirelength
8	5.28	5.02	4.88	4.92
9	5.34	5.11	4.95	5.02
10	5.89	5.19	4.95	5.03
11	6.02	5.33	5.03	5.17
12	6.27	5.69	5.08	5.36
13	6.31	5.98	5.11	5.38

Fig. 9. HHSM-based % classification error for the question of which tool will perform better.

Question 2: How Should The Tool Be Driven? To answer this question, we estimate the parameter subspace in which a commercial tool will return a solution which is feasible with respect to all constraints. We demonstrate the application of our estimation methodology to determine the feasible spaces of maximum skew, maximum

	Max Skew (ps)		Max Delay (ns)		Max Buffer Transition (ps)	
Delay (ns)	*ToolA*	*ToolB*	*ToolA*	*ToolB*	*ToolA*	*ToolB*
0.1	N	N	N	N	N	N
0.5	N	N	N	N	N	N
0.8	10 - 75	N	0.5 - 1.35	N	360 - 675	N
1.0	10 - 75	50 - 175	0.5 - 1.75	1.0 - 3	360 - 675	300 - 775
2.0	10 - 75	50 - 175	0.5 - 2.25	1.0 - 3	360 - X	300 - X
5.0	10 - 105	50 - 300	0.5 - 2.25	1.0 - 5.0	360 -X	300 - X

Fig. 10. Parameter subspaces that satisfy a given delay with realistic testcases.

	Max Skew (ps)		Max Delay (ns)		Max Buffer Transition (ps)	
Skew (ps)	*ToolA*	*ToolB*	*ToolA*	*ToolB*	*ToolA*	*ToolB*
0.5	N	N	N	N	N	N
5	N	N	N	N	N	N
25	10 - 25	25 - 50	1.0 - 1.75	1.5 - 2.50	275 - 450	300 - 475
50	10 - 50	25 - 100	1.0 - 2.0	1.5 - 1.75	275 - 575	300 - X
100	10 -100	40 - 115	1.0 - X	1.5 - X	300 - X	300 - X
200	10 - 100	45 - 115	1.0 - X	1.5 - X	300 - X	300 - X
500	10 - 100	45 - 115	1.0 - X	1.5 - X	300 - X	300 - X

Fig. 11. Parameter subspaces that satisfy a given skew with realistic testcases.

insertion delay, and maximum buffer transition times for a tool to deliver desired values of CTS metrics such as skew and delay. We use exhaustive search based on the following steps to determine the parameter subspaces.

1) Derive two HHSM models, one for *ToolA* and one for *ToolB*.
2) Determine the upper and lower bounds of parameters from the training set.
3) For each parameter, perform binary search within its upper and lower bound values and use the HHSM models for each tool to estimate value of the metric.
4) If the desired value of the metric is within the average error of the HHSM models derived in Step 1, record the smallest and the largest values of the parameter that satisfies the metric. The smallest value is the lower limit, and the largest value is the upper limit of the subspace for the parameter.
5) If the value for the upper limit from Step 4 is equal to the upper bound found from Step 2, then use two values that are 20% and 40% more than the upper bound. If these values also satisfy the metric, then this suggests that the upper limit is unbounded. Use *X* to denote that the upper limit is unbounded. We believe that our training set uses wide ranges of parameter values, and hence that values 20% and 40% above the upper bound are very large.
6) If the metric cannot be satisfied within the upper and lower bounds obtained from Step 2, then use *N* to denote that the metric cannot be satisfied by the tool.

We vary maximum skew from 0.5ps to 350ps, maximum insertion delay from 0.1ns to 5.0ns, and maximum buffer transition time from 150ps to 1000ps. Figures 11 and 10 respectively show the parameter subspaces for which *ToolA* and *ToolB* meet skew and delay requirements on realistic instances. We observe that both *ToolA* and *ToolB* are in general unable to meet tight skew and delay requirements, such as 0.5ps or 1ps for skew and 0.1ns or 0.5ns for delay. Such extreme values help us identify the "field of use" of each tool, that is, values at which the tool fails to meet constraints, or provides the best solution for a specified metric. At skew requirements of 100ps and above, we observe that the upper limits of maximum skew, maximum insertion delay, and maximum buffer transition times are unbounded for both tools. At delay requirements of 2ns and above, we observe that the upper limit of maximum buffer transition times is unbounded.

Question 3: How Wrong Can The Model Guidance Be? We say that a model guidance is wrong when the model predicts that *ToolA* will perform better than *ToolB*, but actual data shows that *ToolB* is better than *ToolA*. When the model guidance is wrong, we quantify *suboptimality* as a percentage, i.e.,

$$SUB = \frac{CTS\ outcome\ difference}{Actual\ better\ tool\ outcome} \times 100 \quad (2)$$

We experimentally study **Question 3** using the following steps.

1) Use our "oracle classifier" to predict which of *ToolA* and *ToolB* will perform better for a given CTS problem instance.
2) Compare post-CTS outcomes of both tools and determine the actual better tool.
3) If the better tools from Steps 1 and 2 mismatch (i.e., the model guidance is wrong), count as a prediction error and calculate suboptimality using Equation (2).

Figure 12 shows the results for power and wirelength when model guidance is wrong. The "SVM" column shows the percentage of occurence of wrong guidance; the "SUB" column shows the suboptimality when wrong guidance occurs. The maximum suboptimality in power (resp. wirelength) is 9.22% (resp. 7.97%) when the tool guidance from our classifier is wrong.

	Power				Wirelength			
	ToolA		*ToolB*		*ToolA*		*ToolB*	
D	SVM	SUB	SVM	SUB	SVM	SUB	SVM	SUB
8	5.38	5.89	5.22	9.08	5.28	4.01	5.32	5.98
9	5.38	9.07	5.24	9.07	5.55	4.87	5.42	7.55
10	5.78	9.2	5.67	9.22	5.61	3.31	5.59	7.87
11	5.8	8.25	6.04	8.96	5.76	7.97	5.88	6.33
12	5.8	6.45	6.22	8.93	6.01	7.23	5.89	5.12
13	5.81	3.12	6.22	8.93	6.03	7.09	5.92	5.35

Fig. 12. Percentage incidence of classification error (SVM), and percentage suboptimality (SUB) of power and wirelength in cases of wrong guidance.

B. Application of Prediction Methodology to a Design Flow

We empirically demonstrate the application of our estimation and classification methodologies to a new CTS problem instance to obtain a specified skew target of $\leq$ 30ps. We use TSMC45GS technology libraries and the new realistic template shown in Figure 13; this template is different from those used to train our models. To obtain a skew target of $\leq$ 30ps, we perform exhaustive search in the parameter subspaces of each tool from Figure 11. We iteratively search for parameter values that can deliver the target skew, as follows

1) From Figure 11, begin with the lower bounds of maximum skew, maximum delay, and maximum buffer transition time in the union of ranges of 25ps and 50ps. For all other parameters, use the minimum values from Table I; the minimum value of M_{CEP} is zero.
2) Perform CTS with both *ToolA* and *ToolB* with the selected parameter values.
3) Check if the post-CTS skew is $\leq$ 30ps for each tool. If yes, exit with success. Otherwise, increase maximum skew, maximum delay and maximum buffer transition by Δ_{skew}, Δ_{delay}, and Δ_{tran} respectively, one at a time, and go back to Step 2.[5]
4) If maximum skew, maximum delay and maximum transition time values all reach their upper bounds, then exit with no feasible solution.

Figure 14 shows that we can meet the target skew in four CTS runs of *ToolA* and five CTS runs of *ToolB*. We observe that if we choose parameter subspaces from results of prediction methodologies (i.e., from Figure 11), the tools deliver post-CTS skew of $\leq$ 30ps and satisfy all constraints.

[5]We set $\Delta_{skew} = 5$ps, $\Delta_{delay} = 0.1$ns, and $\Delta_{tran} = 50$ps.

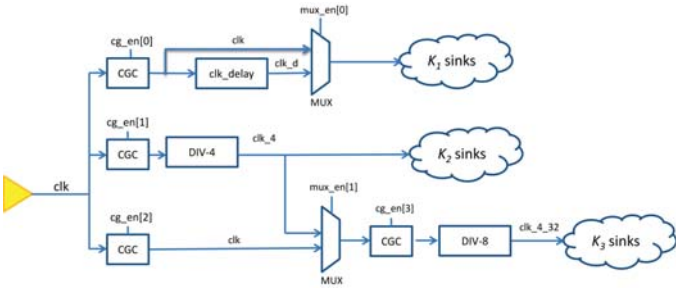

Fig. 13. Realistic template used in CTS problem instance.

Max Skew (ps)	Max Delay (ns)	Max. Buffer Transition (ps)	CTS Tool	Post-CTS Skew (ps)	Number of CTS runs
15	1.1	325	*ToolA*	29.8	4
35	1.6	350	*ToolB*	27.7	5

Fig. 14. Both tools can deliver target skew $\leq$ 30ps when they use parameter subspaces determined using our methodologies.

VI. Conclusions and Future Work

Classical clock tree synthesis remains a critical problem in sub-28nm SoC design because of its substantial power/resource implications. The clock tree can be implemented using a wide variety of design styles and tools, with many parameter and constraint settings that induce a wide range of tool outcomes. In this work, we attack the difficult dimensions of CTS prediction and develop a new HHSM technique to break the curse of multicollinearity at high dimensions. We show that HHSM can reduce the worst-case estimation error by 57% compared to previous metamodeling techniques.

We demonstrate the feasibility of answering three important questions that design teams typically ask, using our classifiers and estimators. We also describe templates of realistic CTS instances in the context of a new testcase generation methodology. Using our realistic testcases, we demonstrate that our prediction methodology can be applied to new CTS problem instances to obtain a desired value of a specific metric when the optimal tool is used.

Our ongoing work focuses on three main directions. First, we continue to explore modern machine learning techniques and model accuracy improvements for high-dimensional, multicollinear parameter spaces. Second, we are pursuing general and scalable methodologies for modeling and characterization of EDA tools, so as to bound solution metrics and efficiently search for optimal parameter choices. An example goal for prediction might be to lower-bound the number of hold buffers inserted by a routing tool as a function of corners, library, deratings, and maximum and minimum paths between flip-flops. Last, we seek to apply our estimation and tool characterization methodologies to reduce time and cost of design space exploration at other stages of IC implementation.

Acknowledgments

We gratefully acknowledge research support from NSF, MARCO/DARPA, Qualcomm and the Semiconductor Research Corporation.

References

[1] D. A. Belsley, "Multicollinearity: Diagnosing Its Presence and Assessing The Potential Damage It Causes Least-Squares Estimation", *National Bureau of Economic Research Working Paper No. 154*, 1976.

[2] G. E. P. Box and M. E. Muller, "A Note on the Generation of Random Normal Deviates", *Annals of Mathematical Statistics* 29(2) (1958), pp. 610-611.

[3] N. Callegari, D. Dramanac, L.-C. Wang and M. S. Abadir, "Classification Rule Learning Using Subgroup Discovery of Cross-Domain Attributes Responsible for Design-Silicon Mismatch", *Proc. DAC*, 2010, pp. 374-379.

[4] S. Chakrabartty and G. Cauwenberghs, "Sub-Microwatt Analog VLSI Trainable Pattern Classifier", *JSSC* 42(5) (2007), pp. 1169-1179.

[5] K. Crombecq, L. De Tommasi, D. Gorissen and T. Dhaene, "A Novel Sequential Design Strategy for Global Surrogate Modeling", *Proc. Winter Simulation Conference*, 2009, pp. 731-742.

[6] S. Chakrabartty and G. Cauwenberghs, "Sub-Microwatt Analog VLSI Trainable Pattern Classifier", *JSSC* 42(5) (2007), pp. 1169-1179.

[7] K. Crombecq, L. De Tommasi, D. Gorissen and T. Dhaene, "A Novel Sequential Design Strategy for Global Surrogate Modeling", *Proc. Winter Simulation Conference*, 2009, pp. 731-742.

[8] F. Dubois, V. Catalano, M. Coppola and F. Petrot, "Accurate On-Chip Router Area Modeling with Kriging Methodology", *Proc. ICCAD*, 2012, pp. 450-457.

[9] T. Goel, R. T. Haftka, W. Shyy and N. V. Queipo "Ensemble of Surrogates", *Structural and Multidisciplinary Optimization* 33(3) (2007), pp. 199-216.

[10] T. Hastie, R. Tibshirani and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, Springer, 2009.

[11] F. J. Hickernell, "A Generalized Discrepancy and Quadrature Error Bound", *Mathematics of Computation* 67(221) (1998), pp. 299-322.

[12] A. A. Ilumoka, "Efficient Prediction of Crosstalk in VLSI Interconnects using Neural Networks", *Proc. EPEPS*, 2000, pp. 87-90.

[13] R. Jin, W. Chen and T. W. Simpson, "Comparative Studies of Metamodeling Techniques under Multiple Modeling Criteria", *Struct. Multidiscip. Optim.* 23 (2001), pp. 1-13.

[14] A. B. Kahng and S. Mantik, "A System for Automatic Recording and Prediction of Design Quality metrics", *Proc. ISQED*, 2001, pp. 81-86.

[15] A. B. Kahng and S. Mantik, "Measurement of Inherent Noise in EDA Tools", *Proc. ISQED*, 2002, pp. 206-211.

[16] A. B. Kahng, B. Lin and K. Samadi, "Improved On-Chip Router Analytical Power and Area Modeling", *Proc. ASPDAC*, 2010, pp. 241-246.

[17] A. B. Kahng, B. Lin and S. Nath, "Enhanced Metamodeling Techniques for High-Dimensional IC Design Estimation Problems", *Proc. DATE*, 2013, pp. 1861-1866.

[18] F. Liu, "A General Framework for Spatial Correlation Modeling in VLSI Design", *Proc. DAC*, 2007, pp. 817-822.

[19] S. N. Lophaven, H. B. Nielsen and J. Sondergaard, "Aspects of the MATLAB Toolbox DACE", *Technical Report IMM-REP-2002-13*, Technical University of Denmark, 2002.

[20] D. F. Morrison, *Multivariate Statistical Methods*, 3rd edition, McGraw-Hill Publishing Company, 1990.

[21] R. Samanta, J. Hu and P. Li, "Discrete Buffer and Wire Sizing for Link-Based Non-Tree Clock Networks", *IEEE Trans. VLSI* 18(7) (2010), pp. 1025-1035.

[22] R.-S. Tsay, "Exact Zero Skew", *Proc. ICCAD*, 1991, pp. 336-339.

[23] V. Vapnik, *The Nature of Statistical Learning Theory*, Springer-Verlag, 1995.

[24] M. B. Yelten, T. Zhu, S. Koziel, P. D. Franzon and M. B. Steer, "Demystifying Surrogate Modeling for Circuits and Systems", *Circuits and Systems Magazine* 12(1) (2012), pp. 45-63.

[25] *ATopTech Aprisa.* www.atoptech.com/aprisa

[26] Broadcom Corporation (networking infrastructure physical design principal engineer), *Personal communication*, September 2012.

[27] ISPD 2010 High Performance Clock Network Synthesis Contest. *http://archive.sigda.org/ispd/contests/10/ispd10cns.html*

[28] *Mentor Graphics Olympus-SoC.* www.mentor.com/products/ic_nanometer_design/place-route/olympus-soc/

[29] Qualcomm Corporation (mobile SoC physical design principal engineer), *Personal communication*, November, 2012.

[30] Samsung Electronics Corporation (System LSI application processor principal engineer), *Personal communication*, February 2012.

[31] *Cadence SOC Encounter User Guide.* http://www.cadence.com

[32] *Synopsys Design Compiler User Guide.* www.synopsys.com/Tools/Implementation/RTLSynthesis/DCUltra/Pages/default.aspx

[33] *Synopsys IC Compiler User Guide. www.synopsys.com/Tools/Implementation/PhysicalImplementation/Pages/ ICCompiler.aspx*

[34] *MATLAB.* http://www.mathworks.com

[35] *ARESLab.* http://www.cs.rtu.lv/jekabsons/regression.html

[36] *RBF2 manual.* http://www.anc.ed.ac.uk/∼mjo/rbf.html